Tecnológico de Costa Rica

High Performance Embedded Systems

Project 3: Optimizing Traditional and Deep Learning Algorithms for Autonomous Driving

Report

Professors

MSc. José Araya Martínez
MSc. Sergio Arriola-Valverde

Students

Sergio Guillén Fernández
Eliécer Mora Alaniz

Second Third of year
2020

SGBM Algorithm Profiling

Given that the result obtained when attempting to profile with valgrind the provided implementation of the SGBM algorithm is that it takes a lot of time just to start the the Graphical User Interface (GUI) in the provided implementation, it was required to port the implementation to a console-based application. After implementing this, it was possible to obtain good results in the profiling.

Part of the most important results are presented in the Illustration 1.

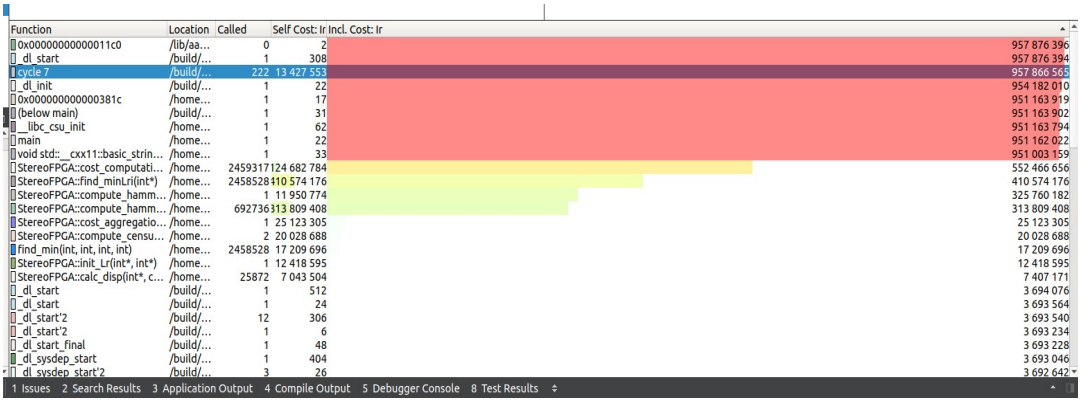

Illustration 1: Results after profiling the SGBM algorithm.

Given this, it was considered that the three most computation-expensive functions are:

- cost_computation.
- find_minLri.
- compute_hamming_distance.

Memory optimization

Once the algorithm was profiled, the next step was to optimize the amount of memory based on some assumptions in the input parameters.

Assuming that the input parameters are restricted to:

- BlockSize: 5 pixels
- P2 : 128
- NumDir: 4
- $C_{máx}$ (p, d): Results from Equation 3

The following results were obtained:

For equation 2:

$$\text{Census Transform Bit depth} = \text{Blocksize} * \text{Blocksize} - 1$$
$$= 5 * 5 - 1$$
$$= 24$$

This allows to reduce the size of the array that holds the census transform from 64 bit elements to 32 bits elements. This is: ct1 and ct2 in the implementation. This implies changing the declaration and implementation of the functions:

- compute_census_transform
- compute_hamming
- compute_hamming_distance

Also, it is possible to reduce the size of the m_u8BlockSize_half to one byte as the maximum value stored here fits in that size.

For equation 3:

$$C_{máx}(p, d) = Log_2 (\text{Census Transform Bit depth})$$
$$= Log_2(24)$$
$$= 4.58$$

Given than this equation provides the result of the bit depth after computing the hamming distance. So, we are changing from a theoretical bit depth of 24 (32 in reality) to one of 4.58 bits (8 in the implementation). This allows to change the size of the elements the application reserves for the hamming distance computation such that they are modified from an integer size (32 bits) to an unsigned integer of 8 bits. Specifically, the type of m_ActiveInitCost was changed from int to uint8_t.

This implied a change in the declaration and implementation of the following functions:

- compute_hamming
- compute_hamming_distance
- init_Lr
- compute_SGM

For equation 4:

$$L \leq C_{máx}(p, d) + P2$$
$$L \leq 4.58 + 128$$
$$L \leq 132.58$$

As this result provides the higher value we'll get for the cost computation, we conclude the size of the reserved elements for the cost computation can be reduced from an integer to an integer of 8 bits. Concretely, the type of the pointer m_ActiveLrCost is set to an int8_t.

All the related function were updated to use this type.

For equation 6:

$$S \leq \text{Num. Dir} * (C_{máx}(p, d) + P2)$$
$$S \leq 4 * 132.58$$
$$S \leq 530.32$$

Finally, this compute value allows us to change the type used for the aggregate in the cost function such that it is changed from an integer to an unsigned integer of 16 bits.

Multi-threading

The initial execution time for the cost_computation function is around 12.798 s, for the find_minLri it is around 242 us. This time is really small, however, as can be seen in Illustration 1, it is called a lot of times and that's why it was selected. Finally, the function compute_hamming takes around 2.386 s without any optimization.

The results obtained after attempting to optimize the mentioned functions in the profiling section of this report is summarized in the table 1.

Function	Initial execution time (s)	Optimized execution time (s)
cost_computation	12.798	3.33
find_minLri	242*E-6	242*E-6
compute_hamming_distance	2.386	2.378

Table 1. Summary of the initial and optimized execution times

As seen in the table, there are some functions that were not optimized. It was attempted to reduce the execution time, however, the obtained result is that the execution time was increasing as the threads augmented. In the case of the hamming distance, the result was similar but the execution time did not increase as much as with the find_minLri function.

Conclussions

As it was seen in the profiling section, having a GUI application causes inconveniences while attempting to profile the application with valgrind. A console-like porting was implemented to be able to profile the application and determine the most computation-expensive functions in the implementation. The issue with the GUI application is related with all the functions the GUI needs to be constantly calling to maintain the interface in correct functioning. The profiling allowed to identify that the cost_computation, find_minLri and compute_hamming_distance functions are the most computation consuming functions in the current implementation of the SGBM algorithm.

On the other side, the memory foot-print of the application was reduced by analyzing the algorithm, its input data and the required intermediate computations. This lead to a considerable reduction of the required memory. Given we are in an embedded system, this represents a good improvement in the application.

Once the memory usage was reduced, a multi-threading optimization was performed in the application to reduce the execution time and exploit in a better way the computation power of the device in use. Specifically, it was possible to reduce the computation time for the cost computation function in around the 75% of the initial execution time. This represents an important improvement in the algorithm because this function is the most computation expensive function in the implementation.

It was expected to implement NEON intrinsics in the targeted functions but it was not possible to meet the deadline for this project and all the required steps.

Given that, there is an important opportunity to improve this project by continuing with the NEON intrinsics implementation as it is expected this will cause a more important improvement in the computation time of this algorithm.